DETAILED DESCRIPTION

[Detailed Description of the Invention]
[0001]
[The technical field to which invention belongs] Concerning the microprocessor containing a floating-point unit and a central processing unit, especially the pipeline control technology in it, this invention is applied to the microprocessor of RISC format, and relates to effective technology.
[0002]
[Description of the Prior Art] A certain RISC (reduced instruction-set computer) microprocessor is equipped with FPU (floating-point unit). A floating-point unit is a circuit which performs floating point calculation. RISC is relatively simple and is the computer architecture which decreased the complexity of a microprocessor using the instruction of a fixed size. Almost all instructions in RISC architecture operate an operand using a general-purpose register, and store a result in a register. These registers are loaded from memory and, typically, the contents of the register are reused in program execution. Almost all RISC architecture has 16 or a general-purpose register beyond it.
[0003]
[Problem(s) to be Solved by the Invention] A typical RISC microprocessor has the capacity to execute an instruction on a pipeline. There are many problems in adjusting actuation of many functional units (FPU pipeline who performs the CPU pipeline who performs integer processing of CPU etc., and floating point processing). If two units (the unit which controls a CPU pipeline, and unit which controls an FPU pipeline) in such equipment shared the resource for pipeline control, what synchronizing actuation of two pipelines will play the big role to solution of the problem for was clarified by this invention person. That is, the macro processor containing FPU and CPU is considered. Conventionally, each number of stages (the number of pipeline stages) of an FPU pipeline and a CPU pipeline is different. Since a floating point arithmetic requires the operation time compared with an integer arithmetic, some which assigned three steps of pipeline stages are in an activation stage. When the number of stages of an FPU pipeline and a CPU pipeline is different, it is necessary to constitute separately the decode logic for performing flow controls, such as a stall and a bypass, etc. for every pipeline of both CPU and FPU. For example, if the bypass between the pipelines for every instruction is considered as an example, when producing a data conflict like a register conflict with an instruction [before and after], Although an operand can be obtained without the instruction of the back concerned waiting for the completion of activation of a previous instruction by giving the result of the activation stage directly to the activation stage of a next instruction in a bypass path when a pre- instruction completes an operation It must be determined from which stage depending on the number of stages of a pipeline stage which stage should be made to bypass. If the length (pipeline number of stages) of an FPU pipeline and a CPU pipeline is different at this time, it is necessary for the resource for flow controls, such as by-pass control, to also almost change an individual exception between a CPU pipeline and an FPU pipeline, and complicated hardware will be needed even if it faces the data exchange between an FPU pipeline and a CPU pipeline. this invention person found out about these points.
[0004] Another problem is maintaining exceptional semantics correctly. When an exception or an interrupt occurs, processing an exception or interruption correctly in the pipelined architecture or multi-function unit architecture (** which does not disturb program execution sequence) means what saves the condition of a machine so that it may become the same as the program and accuracy which are continuously performed on architecture. Though the instruction which publishes directions to a junction unit even if maintains the sequence of a program strictly, since the execution times of an instruction differ, it is confused with the functional unit from which the execution sequence of an instruction

differed. For example, it sets to the processor which has a CPU pipeline and an FPU pipeline. When processing the instruction with more execution cycles than usual [like a floating-point-divide instruction] on an FPU pipeline, the running state of the instruction which continues after the floating-point-divide instruction concerned when an exception like zero division occurs in the processing -- depending on how If it originates in exception generating and the floating-point-divide instruction concerned is rerun, instruction-execution sequence may disturb the instruction-execution sequence specified by the program, and control may become complicated.

[0005] The effective means for interrupting correctly in above semantics in the pipelined processor is discussed by the IEEE computer processing as advanced technology, and page [562-573rd] "strict interruption in the pipelined processor" ("Implementing Precise Interrupts in Pipelined Processor"IEEE Transaction on Computers, pp.562-573, and May 1988). Most processors which have many functional units with which current was pipelined are based on the technology mentioned there.

[0006] A certain thing of these technology needs the addition of a register file, and needs the complicated logic for control. In order to share and synchronize a resource, the hardware of tag matching is needed like the still more complicated internal data bus combined with the share resource. The register scoreboard for identifying the conflict (contention) of a register resource and canceling is used for other technology. In short, such technology requires the addition of chip area and is not suitable for the processor of the low price which plans en BEDDETTO application (device inclusion control).

[0007] The purpose of this invention is to offer the microprocessor which can synchronize a floating point pipeline and an integer pipeline.

[0008] Furthermore, this invention is to offer the microprocessor which can reduce the resource for pipeline control on the occasion of the above-mentioned synchronization.

[0009] Another purpose of this invention is to offer the data-processing method that the data exchange between CPU and FPU can be easily performed at a high speed.

[0010] Still more nearly another purpose of this invention is to offer the data-processing method which can raise the throughput to conditional branching which continues after a floating-point-compare instruction.

[0011] The purpose of others of this invention is to offer the data-processing method which can synchronize a floating point pipeline and an integer pipeline also in a floating-point-divide instruction.

[0012] The other purposes and the new feature will become clear from description and the accompanying drawing of this specification along [said] this invention.

[0013]

[Means for Solving the Problem] It will be as follows if an outline of a typical thing is briefly explained among invention indicated in this application.

[0014] namely, -- A microprocessor (110) executes an instruction using an integer pipeline (CPU pipeline) who performs a floating point pipeline (FPU pipeline) who performs floating point processing, integer count, and memory address actuation. That is, on the occasion of an instruction execution, an FPU pipeline and a CPU pipeline are advanced in juxtaposition. By making the same fundamentally a pipeline number of stages (a pipeline's length) of a floating point pipeline (210) and a CPU pipeline (214), share-ization of a resource for pipeline controls, such as a flow control of a CPU pipeline and an FPU pipeline, is attained. A floating point pipeline and an integer pipeline are made easy to synchronize by this. When you need a frieze and a stall to a pipeline, and a pipeline's one side controls a stall or a frieze to bring both pipelines an effect of a stall and a frieze, it attains synchronization of an FPU pipeline and a CPU pipeline.

[0015] Floating point processing that an exception is long is also kept exact. An exact exception is attained by what (it enables) a busy signal is asserted for on an activation stage of the beginning of an FPU pipeline stage when a floating-point-divide instruction which is a floating point instruction of the point in instruction-execution sequence which a program specifies goes into an activation stage of an FPU pipeline's beginning. When a next floating point instruction is decoded by FPU pipeline before completing activation which spent time amount for two or more stages for which a floating point instruction of said point repeats and uses an activation stage of an FPU pipeline's beginning, the stall of the pipeline of both sides of CPU and FPU after it is carried out.

[0016] A means by this invention is explained further.

[0017] Said floating point pipeline's pipeline number of stages and said integer pipeline's pipeline number of stages are made equal, and a microprocessor containing the floating point and a central processing unit which execute an instruction using an integer pipeline (214) who performs a floating point pipeline (210) who performs floating point processing, integer processing, and memory addressing

processing changes. If a number of stages of both pipelines (210,214) is equal, both pipelines can be made to share a resource for a flow control which controls a stall and a frieze. This enables contraction of a chip area. Sharing of a resource for a flow control makes synchronization of both pipelines easy.
[0018] Said integer pipeline's decoding stage can generate a stall signal for a floating point pipeline and an integer pipeline.
[0019] In a microprocessor containing a floating-point unit and a central processing unit, in a viewpoint of a data-processing method of executing an instruction using an integer pipeline who performs a floating point pipeline who performs floating point processing, integer processing, and memory addressing processing, said floating point pipeline's pipeline number of stages is made equal to said integer pipeline's pipeline number of stages, and said floating point pipeline and integer pipeline have a common instruction fetch stage. At this time, there is a data-processing method by the 1st instruction which transmits data to a register (FPUL) of a floating-point unit from a register of a central processing unit as an example of synchronization with an FPU pipeline and a CPU pipeline when executing a certain instruction. The data-processing method of executing this 1st instruction includes processing which fetches this instruction on said instruction fetch stage, processing whose integer pipeline decodes said 1st instruction and outputs a value of a register of a central processing unit to a bus, and the processing which write a value of said bus in a register of a floating decimal unit synchronous by a floating point pipeline decoding said 1st instruction with said processing to output.
[0020] Moreover, a data-processing method by the 2nd instruction which transmits data to a register of a central processing unit from a register of a floating-point unit can be mentioned as another example of the above-mentioned synchronization. The data-processing method of executing this 2nd instruction includes processing which fetches the 2nd instruction concerned on said instruction fetch stage, processing whose floating point pipeline decodes said 2nd instruction, and outputs a value of a register (FPUL) of a floating-point unit to a bus, and the processing which write a value of said bus in a register of a central processing unit synchronous by an integer pipeline decoding said 2nd instruction with said processing to output.
[0021] Said floating point pipeline's pipeline number of stages is made equal to said integer pipeline's pipeline number of stages. In said data-processing method in case said floating point pipeline and integer pipeline have a common instruction fetch stage Said floating point pipeline further A decoding stage, the 1st activation stage, When it has an activation stage and a write back stage in the 2nd and said integer pipeline has a decoding stage, an activation stage, a memory access stage, and a write back stage further, A planned data-processing method raising a throughput of conditional branching by conditional-branching instruction which follows a floating-point-compare instruction Processing whose floating point pipeline generates T bits on the 1st activation stage to a floating-point-compare instruction which generates T bits of a logical value according to coincidence/inequality of a comparison result, Processing whose integer pipeline refers to said T bits on the activation stage to a conditional-branching instruction which makes a value of T bits branch condition is included. When a conditional-branching instruction is arranged immediately after a floating-point-compare instruction by this, a stall is unnecessary entirely.
[0022] Said floating point pipeline's pipeline number of stages is made equal to said integer pipeline's pipeline number of stages. In a data-processing method in case said floating point pipeline and integer pipeline have a common instruction fetch stage Said floating point pipeline further as well as the above-mentioned A decoding stage, When it has an activation stage and a write back stage in the 1st activation stage and the 2nd and said integer pipeline has a decoding stage, an activation stage, a memory access stage, and a write back stage further, A data-processing method which maintains synchronization with an FPU pipeline and a CPU pipeline by floating-point-divide instruction Processing which fetches a floating-point-divide instruction which repeats the 1st activation stage two or more times on an instruction fetch stage, In processing which enables a busy signal at a period of the 1st activation stage in said floating-point-divide instruction by which the fetch was carried out, and a period which said busy signal is enabling Processing which both a pipeline stage after the 1st [of an instruction of consecutiveness] activation stage using a floating point pipeline and an integer pipeline's stage are synchronized, and carries out a stall is included.
[0023]
[Embodiment of the Invention] In the suitable example of this invention, a microprocessor uses the integer pipeline (CPU pipeline) 214 (refer to drawing 2), in order to use the floating point pipeline (FPU pipeline) 210 (refer to drawing 2) in order to perform floating point processing and to perform an integer, memory addressing processing, etc. so that it may be explained below. It goes on in [the FPU

.pipeline 210 and the CPU pipeline 214] juxtaposition, and synchronization is performed in the process. The pipeline number of stages (a pipeline's number of stages) of the FPU pipeline 210 and the CPU pipeline 214 is made equal to mutual. By this, the resource for the pipeline controls for the flow control of the CPU pipeline 214 and the FPU pipeline 210 etc. can share [an FPU pipeline and a CPU pipeline] almost now. Sharing of this pipeline resource makes easy synchronization with the CPU pipeline 214 and the FPU pipeline 210.

[0024] The FPU pipeline 210 is synchronized by the CPU pipeline 214 in some respects. FPU and the CPU pipeline 210,214 are the times of FPU114 and CPU118 sharing [an instruction like loading / store / restoration instruction of the floating point] a resource (for example, communication register) by synchronizing mutually. FPU and the CPU pipeline 210,214 are synchronized when they perform the data exchange using a communication register FPUL. Moreover, a pipeline's synchronization is performed also in the floating-point-compare instruction with which for example, CPU branch instruction follows [the procedure of an instruction]. These two pipelines' 210,214 synchronization is effective also in maintaining an exact exception so that it may explain below further.

[0025] Drawing 1 shows the block diagram of the processor concerning a suitable example of this invention. A processor 110 has a floating-point unit (FPU) 114. Furthermore, a processor 110 is equipped with the central processing unit (CPU) 118 which can operate an integer. Let a processor 110 be the 32-bit RISC architecture equipped with the 16-bit fixed-length floating-point-instruction set. Details of the 16-bit instruction for CPU118 are given to 1995 ("SH3:High Code Density, Low Power", and IEEE Micro, pp.11-19 and December 1995) in the 11th [-] month and a paper, i.e., the "SH3, high code density, and the low power" IEEEMicro, and December [19 or]. Said CPU118 is combined with FPU114 via the 32-bit data bus 122. The integral accumulation multiplication (sum-of-products operation) unit (IMAC) 120 is combined with the data bus 122. The interface signal between the circuits of drawing 1 has omitted illustration except interface signal 124,125 between CPU118 and FPU114. Said CPU118 is combined with memory management unit equipment (TLB controller) 134 via the 32-bit logic address bus 126. TLB expresses the foresight mold translation lookaside buffer. The TLB controller 134 controls TLB138 of the mixture mold of an instruction and data. The TLB138 is connected to the cache controller 142 through the 32-bit physics address bus 130. Said cache controller 142 controls the cache memory 146 which made an instruction and data intermingled. Cache memory 146 is combined with CPU118 and FPU114 through the 32-bit data bus 122.

[0026] A microprocessor 110 is further equipped with the serial communication interface 152 connected to the circumference data bus 150 and the circumference address bus 151, and timer 153 grade, and the interface of these circumference data bus 150 and the circumference address bus 151 is carried out to said data bus 122 and the physical address bus 130 through the bus state controller 154. The interruption controller 155 performs mediation of the interrupt request from inside and outside etc. The bus interface with the chip exterior is performed by the external bus interface circuitry 156. If based on the name shown in this explanation and drawing 1 , the function of the other circuits shown in drawing 1 will be naturally understood for this contractor.

[0027] The function of a processor 110 will be understood by the following explanation. FPU114 needs data and an instruction for floating point processing. In this example, FPU114 does not have the addressing capacity of the memory for storing data in cache memory or gaining data with it. This removes the need for the memory addressing circuit of FPU, and saves a chip area. Instead, CPU118 carries out addressing of the cache memory instead of FPU114. CPU118 not only performs the fetch of data from memory, but fetches all instructions containing the floating point instruction for FPU114 from memory for FPU114. Furthermore, data transfer between FPU114 and CPU118 is instead performed at a high speed through a register like a FPUL register again, without being accompanied by memory access so that details may be explained by the following.

[0028] In order to gain data or an instruction, CPU118 calculates the logical address (virtual address), sends the logical address to memory management equipment 134 through the 32-bit logic address bus 126, and requires the data from memory 116, or an instruction.

[0029] If the corresponding physical address is not already stored in TLB138, a TLB mistake will occur and the TLB controller 134 will begin the program sequence for changing the logical address into a physical address using additional mapping information. Memory management equipment 134 stores a physical address in TLB138 then. Since CPU118 may require the data of the again same address range, TLB134 stores the address as a new entry for future use. TLB138 sends a physical address to a cache controller 142 through the 32-bit physics address bus 130. A cache controller 142 is directed to the cache memory 146 in which data and an instruction are intermingled so that data or an instruction may

. be put on the 32-bit data bus 122. Supposing the demanded address is not available on cache memory 146, a cache mistake occurs, and processing of CPU118 and FPU114 will be frozen by applying a cache mistake signal until the fetch of the demanded information is carried out to cache memory from external memory. here -- a frieze -- an instruction -- generally it has a different concept from the stall which stops a pipeline according to a condition like a memory conflict which originates in the array of its situation and is produced, or a register conflict. That is, a frieze means a certain thing [stopping during the period] for a pipeline according to the factor which does not originate in the array (execution sequence of an instruction) of an instruction, for example, a cache mistake, a TLB mistake, etc. A stall and a frieze are unchanging for it being one means to cope with a hazard (failure).

[0030] An instruction is picked up by CPU118 like FPU114 for decoding. Data is available on the 32 bit data bus of community shared by CPU118 and FPU114. FPU114 does not have the capacity to perform memory addressing for data and an instruction fetch.

[0031] The data exchange between CPU118 and FPU114 is performed through the communication register FPUL of dedication. As mentioned above, in other RISC processors, the data exchange between CPU118 and FPU114 happens through transfer memory (transfer through memory) like cache memory 146. The transfer through a register like a FPUL register is quick compared with the transfer through cache memory 146.

[0032] The list of CPU instructions relevant to a floating point instruction and a floating point instruction is shown in drawing 20 . Details are described by drawing 21 - drawing 52 , and these instructions include C description of an instruction. The underline without the importance related to the semantics of an instruction is shown in these instructions enumerated by drawing 21 - drawing 52 .

[0033] To drawing 2 , the pipeline who uses for FPU114 and CPU118 executing an instruction is illustrated. The FPU pipeline 210 and the CPU pipeline 214 share the single instruction fetch stage 218. Furthermore, the FPU pipeline 210 has further 4 of the decoding stage (Df) 222, the 1st activation stage (E1) 226, the 2nd activation stage (E2) 230, and the write back stage (Sf) 210 stages. CPU -- a pipeline -- 214 -- decoding -- a stage -- (-- Di --) -- 238 -- activation -- a stage -- (-- E --) -- 242 -- memory access -- a stage (A246) -- and -- write back -- a stage -- (-- Si --) -- 250 -- further -- four -- stages -- having . The Di stage 238 generates the signal for a bypass, a stall, and cancellation to an FPU pipeline stage as shown by the arrow head 255 of drawing 2 . Through the signal pass 255, the Di stage 238 supplies a bypass signal to E1 and E2 stage 226,230, supplies a cancellation signal to all the FPU stages 222,226,230,234 and share I stages 218, and supplies the signal relevant to a stall to Df stage, and supplies a stall signal to E1 stage. These signals are explained further below.

[0034] The instruction which can be used on a data bus 122 is gained by the instruction fetch stage 218 at first. Both decoding stages Df222 and Di238 decode the instruction by which the fetch was carried out on the instruction fetch stage. The 1st phase of a decoding stage includes identifying a CPU instruction or an FPU instruction. An FPU instruction is identified by F (hexadecimal) in 4 bits of high orders of an instruction. If an instruction is not a floating point type, the Df stage 222 will not decode an instruction any more. Similarly, the Di stage 238 does not decode a floating point instruction completely. The Di stage 238 does not carry out decoding a floating point instruction, in order to identify the floating point function which should be performed. This brings about contraction to the complexity of hardware and is an important thing. If only a single decoding stage is used, the signal demanded in order to control the data path of FPU must cross CPU118 to FPU114, and will make a chip area increased. When the instruction by which the fetch was carried out is a floating point instruction, 210EFPU pipeline 1 stage 226 begins to execute an instruction. 210EFPU pipeline 2 stage 230 completes activation of the floating point instruction. Depending on the requirement of an instruction, the FPU pipeline's 210 Sf stage can store the activation result of an instruction in a floating-point register.

[0035] When the instruction by which the fetch was carried out is a CPU instruction like the instruction for operating an integer, the CPU pipeline's 214 E stage 242 executes the instruction concerned. The CPU pipeline's 214 A stage 246 accesses cache memory 146, when it is required with the specific instruction currently executed. Finally, the CPU pipeline's 214 Si stage 250 can write an instruction-execution result in one of the CPU registers 410. The instruction which requires that only one of the FPU pipeline 210 or the CPU pipelines 214 should be used for an advantageous thing will be called the instruction which only passes it about the pipeline who is not used. For example, when the instruction fetch stage 218 fetches an integer add instruction, the CPU pipeline 214 performs integer addition on the activation stage 242, and stores the activation result in a register on the Si stage 250. However, when an integer add instruction is decoded, the FPU pipeline's 210 Df stage 222 makes E1 stage 226 pass an

,integer add instruction. An integer add instruction continues passing through the FPU pipeline's 210 remaining stages between next two or more clock cycles. Similarly, when the instruction by which the fetch was carried out is a pure floating point instruction, said Di stage 238 makes the E stage 242 pass a floating point instruction, and only passes the CPU pipeline's 214 remaining stages between next two or more cycles. The above-mentioned passage means that a pipeline stage without substantial operation only passes the pipeline in the CPU pipeline and FPU pipeline who have the relation of the front reverse side here.

[0036] Here, the meaning of constituting a CPU pipeline and an FPU pipeline so that it may be illustrated by drawing 2 is explained. That is, the pipeline number of stages (a pipeline's number of stages) of the FPU pipeline 210 and the CPU pipeline 214 is made equal to mutual. By this, the resource for the pipeline controls for the flow control of the CPU pipeline 214 and the FPU pipeline 210 etc. can share [an FPU pipeline and a CPU pipeline] almost now. The physical circuit scale for pipeline control is reducible with sharing of this pipeline resource. For example, if the case where a register conflict as shown in drawing 18 is produced is made into an example When the destination register (R1) of a previous instruction "Add R0, R12" is a source register (R1) of a next instruction "Add R1, R2", If it is not after the result of an operation is obtained by the destination register (R1) of a previous instruction, it is meaningless for a next instruction to refer to the value of the register (R1). Therefore, although illustration is not carried out, the stall of the pipeline P2 can be carried out until the result of an operation is obtained by the register R1 on a pipeline's P1 write back Si stage. It is not necessary to perform a stall by bypassing and passing the result of an operation calculated on a pipeline's P1 activation stage E to a pipeline's P2 activation stage E at this time. As shown by pipelines P3-P5, when similarly the instruction "Add R10, R11" which produces a data conflict about a register R11, and another instruction between "Add R11, R12" intervene It is not necessary to perform a stall by bypassing and passing the result of an operation calculated on a pipeline's P3 activation stage E to a pipeline's P5 activation stage E on the memory access stage A. Depending on the length (number of stages) of a pipeline stage for the period which performs an above-mentioned stall, and control to which stage to perform the above-mentioned bypass from which stage will be understood. Usually, control of a stall is controlled by the pipeline side by whom a stall should be done, and by-pass control is controlled by the receptacle side of the data bypassed. If it does so, when the number of stages of an FPU pipeline and a CPU pipeline is different, about the logical circuit for performing the resource which performs control for the data exchange between pipelines who are represented by the bypass, a stall, a frieze, cancellation, etc., i.e., a pipeline's flow control etc., the control line for a bypass, etc., between a CPU pipeline and an FPU pipeline, it turns an individual exception, and if there is nothing, it will hardly become. The pipeline number of stages of the FPU pipeline 210 and the CPU pipeline 214 can be made equal to mutual, and the example shown in the gestalt of this operation can be made to almost share the resource for the pipeline controls for the flow control of the CPU pipeline 214 and the FPU pipeline 210 etc. on an FPU pipeline and a CPU pipeline so that it may be illustrated by drawing 2 . Drawing 19 shows notionally the share condition of the logic for the resource which controls a CPU pipeline and an FPU pipeline, especially flow processing, the logic of the flow processing for a CPU instruction is shared by the flow processing for an FPU instruction, and the addition logic for an FPU instruction is only slightly added to the logic of the flow processing. Let this addition logic be the flag with which the object of a flow control shows an FPU pipeline or a CPU pipeline.

[0037] Hereafter, various synchronization's example of a CPU pipeline and an FPU pipeline is explained.

[0038] A certain instruction requires the data transfer between the FPU pipeline 210 and the CPU pipeline 214. As an example of such an instruction, the load instruction to the communication register (FPUL) by CPU (LDSRm, FPUL), i.e., a CPU load communication register instruction, occurs. This instruction is shown by drawing 3 which shows the data transfer timing between these pipelines 210,214 to the FPU pipeline 210 and CPU pipeline 214 list. A pipeline's configuration is as drawing 2 having explained, and is **. Note that each stage of a pipeline synchronizes with the single clock cycle of the phase (the phase of two clocks is not shown are clear) of one clock 308 again. That is, one pipeline stage is performed in 1 cycle of a clock 308. FPUL is a register contained in FPU114. FPU110 and CPU118 share this register FPUL on data processing. The load instruction to the communication register (FPUL) by CPU is a CPU instruction. However, all instructions are decoded as mentioned above on both decoding stages 222,238 in FPU and the CPU pipeline 210,214. Therefore, since, as for the Df stage 222, FPU114 will control access to Register FPUL as soon as it decodes the load instruction to the communication register (FPUL) by CPU, it is recognized as the FPU pipeline 210 being operated

substantially. At first, the instruction which CPU loads to a communication register (FPUL) is executed by the CPU pipeline's 214 E stage. 210EFPU pipeline 1 stage 226 makes coincidence pass an instruction, without operating in any way so that it may be shown by the alphabetic character (T) 310. In other words, the instruction which CPU loads to a communication register (FPUL) will only be passed to E1 stage.

[0039] Here, another explanation relevant to E1 stage is given. Activation takes 1 cycle to each stage of a pipeline 210,214. However, there is a special case where an instruction says that much time amount is spent on a pipeline stage rather than 1 cycle. In that case, an instruction will repeat the specific pipeline stage. For example, a floating-point-divide instruction (FDIV) as shown in drawing 28 and drawing 29 has rhe TENSHI (latency time) of 13 cycle. Here, rhe TENSHI is the scale of the sum total of the number of cycles which an instruction spends on the FPU pipeline's 210 activation stage 226,230. A floating point instruction spends 1 cycle on E2 stage 230. By it, a floating-point-divide instruction spends 12 cycles on E1 stage so that clearly. The pitch (Pitch) of a floating-point-divide instruction is shown in drawing 28 , and it is the scale of the clock cycle before being able to carry out the activation initiation of the instruction which continues after the present instruction on a pipeline generally. For example, since the pitch of a floating-point-divide instruction is equal to 12 cycles, the next instruction which continues after a floating-point-divide instruction can be executed after 12 cycles. As for the value 12 of a pitch, a floating-point-divide instruction shows 12 cycle expense and ** on E1 stage 226. Therefore, before it goes into E1 stage 226, if the following floating point instruction is 12 clock-cycle **********, there is. [no]

[0040] if it returns to the example of the load instruction to the communication register (FPUL) shown in drawing 3 , since the same instruction will flow to the both sides of FPU and the CPU pipelines 210 and 214, the resource of E1 stage 226 is the same as the number of cycles which the CPU pipeline's 214 E stage performs -- it ******. Next, the CPU pipeline's 214 A stage 246 puts the contents of the register "Rm" referred to with an LDS instruction (CPU load communication register instruction) on a data bus 122. Since this instruction is not an instruction which requires access of cache memory, the A stage 246 only loads the data of the register Rm of the CPU register file 410 to a data bus. That is, the stage A(T') 314 means that memory access does not break out.

[0041] When the CPU pipeline has put the contents of the register "Rm" on the data bus 122, the FPU pipeline's 210 E2 stage 230 passes an instruction, without being accompanied by any actuation so that it may be shown by the alphabetic character "T." A CPU pipeline's A stage 246 enables it to use the contents of the period when the data-ready time amount 318 was restricted, and the register Rm on a data bus 122. With data-ready time amount, a data bus is made busy time amount by the data transfer relevant to a CPU load communication register instruction. During said period 318 when the data on a data bus 122 is available, the FPU pipeline's 210 write back stage Sf234 gains the data on a data bus 122, and stores the data in Register FPUL.

[0042] The number of stages of the pipeline stage of the CPU pipeline 214 and the FPU pipeline 210 is the same so that clearly also from this example. Therefore, if the same sequence as the writing to memory is taken for CPU118 (there is no memory access a line crack in fact), the FPU pipeline 210 can incorporate the data to a FPUL register on Stage Sf.

[0043] CPU store instruction "STS FPUL, Rn" is executed by two pipelines 210 and 214 like the above, as shown in drawing 4 . The CPU store instruction copies the contents of the register (FPUL) to CPU general-purpose register Rn. In the case of this CPU store instruction, FPU114 controls the period 322 which makes the contents of the register FPUL available on a data bus 122. That is, an FPU pipeline puts the contents of the FPUL register only for the period of 322 on a data bus, and the CPU pipeline 214 incorporates this data to Register Rn on Stage Si.

[0044] Drawing 5 shows the FPU pipeline's 210 still more detailed circuit 406. The latch 414,418,422,426 of the master slave format included on the FPU pipeline stage 222,226,230,234, respectively is shown in the circuit of drawing 5 . Synchronizing with the latch 414,418,422,426 corresponding to them for any of the falling edge of the clock signal 408 of a plane 1 eye, or the falling edge of the clock 410 of eye two phases their being, the FPU pipeline stage 222,226,230,234 stores those outputs. Moreover, the bypass signal 428 described below is shown. The function of the FPU pipeline circuit 406 is boiled immediately, and is shown by the example described below. In the example of a CPU load communication register instruction, the FPU decoding stage (Df) 222 performs control which acquires the contents of the register Rm from a data bus 122 by asserting a selection signal on the selection-signal path 430 of a multiplexer 434 so that data may be loaded to a FPUL register.

[0045] Drawing 6 shows an example contemporary with the pipelines 210 and 214 in the activation procedure of two instructions, a CPU instruction and an FPU instruction. Especially drawing 6 shows the conversion command 510 (refer to drawing 41 and drawing 42) through which it passes to the floating point cut-off and integer for which the above-mentioned CPU store instruction will follow. In drawing 6 , in order to simplify illustration, each pipeline stage of the FPU pipeline 210 in one instruction and the CPU pipeline 214 is unified to one continuous block, and is illustrated. Therefore, for example, it is shown with the alphabetic character "D" which shows Df222 and Di238 instead of two decoding stages being shown. In the display of drawing 6 , time amount passes on the right since the left so that it may be shown by one phase of a clock signal 514 (two phases of a clock signal are not shown since it is easy). Since the condition that an instruction only passes through a suitable stage is expressed [else], this pipeline's transcription is standard in this technical field except for the point that the alphabetic character "T" is inserted into a parenthesis, in assignment of a pipeline stage. For example, while the activation stage (E1) 226 of the FPU pipeline 210 of a floating-point unit is executing the floating point cut-off instruction 510, the CPU pipeline's 214 activation stage (E) 242 only passes an instruction, without operating in any way so that it may be shown by the alphabetic character "T."
[0046] If an FPU instruction "FTRC Frm, FPUL" and CPU instruction "STS FPUL, Rn" of drawing 6 are compared, a data conflict (register conflict) will be produced about FPUL so that clearly. In order to prevent generating of the stall by the data conflict (i.e., while the CPU pipeline 214 is executing store instruction, in order to prevent the stall in the CPU pipeline 214), the CPU decoding stage 238 in said instruction "FTRC" asserts the bypass signal 522 to the bypass signal path 255. Assertion of the bypass signal 522 makes available the output of E2 stage 230 of the FPU pipeline 210 who kicks to the input of E2 stage 230 of the store instruction "STS" concerned at said instruction "FTRC", when preparation whose E2 stage of an instruction "STS" are and executes store instruction is completed after E2 stage 230 of an instruction "FTRC" carried out the completion of activation of the floating point cut-off instruction 510.
[0047] The pipeline circuit of drawing 5 attains the above-mentioned bypass of data. That is, the output of the latch 422 of E2 stage becomes available in the bypass path 438. The bypass path 438 is the input of a multiplexer 442. 428 chooses the output of the latch 422 ofEbypass signal 2 from the CPU decoding stage 238 stage, and it can be recycled by it through E2 stage 230. The time amount to which the result of the floating point cut-off instruction 510 becomes available through the output latch 422 of E2 stage 230 is shown by 526 in drawing 6 . The bypass signal 428 from the Di stage 238 of CPU becomes active to the suitable time amount which makes a data transfer possible through a multiplexer 422. As already explained, in order to transmit FPU114 to the CPU register Rn in the above-mentioned store instruction "STS", the period 322 of the place where the data on a data bus 122 is available is controlled.
[0048] synchronization of the FPU pipeline 210 and the CPU pipeline 214 is mainly attained, when either of these pipelines 210,214 that bring about the effect of a stall and a frieze on both pipelines has the control function of a stall and Fries. As shown in drawing 2 , a CPU pipeline's decoding stage (Di) 238 carries out the stall of the FPU pipeline through the stall signal data path shown by the arrow head 255. A pipeline stall happens, while the contents of the register are used with the next instruction, when the first instruction carries out a light to a register between instruction sequence. The stall of the next instruction concerned is carried out until a register is updated with said first instruction, in order to make it not make the contents of the register which is not updated use for said next instruction. In the mode of the operation explained here, it was generated by bypassing the output of the activation stage (E) 242 of one instruction to the input of the activation stage (E) 242 of the next instruction, and such a stall does not have ** and is come. A similar bypass exists in an FPU pipe. NOP (non, - operation) is generated between stalls by the pipeline stage (for example, the E stage 242 or E1 stage 226) by which a stall should be carried out.
[0049] Thus, a bypass can be used for making it not generate a stall when producing a register conflict. At this time, the CPU decoding stage Di performs control of a bypass so that clearly also from drawing 2 . That is, since the number of stages of the pipeline stage of an FPU pipeline and a CPU pipeline is fundamentally made the same, the logic of the flow control which controls a stall, Fries, and a bypass is sharable between a CPU pipeline and an FPU pipeline. In this semantics, the resource for the flow control of an FPU pipeline and a CPU pipeline is sharable by both pipelines. Although the resource for pipeline control is sharable by both pipelines 210 and 214 therefore, FPU and CPU do not need to have the hardware of a flow control separately, a CPU pipeline's flow control logic can be diverted also to an FPU pipeline's flow control, and, thereby, it becomes easy to synchronize a CPU pipeline and an FPU pipeline.

[0050] Next, the circuit relevant to additional stall conditions is illustrated. Drawing 7 is a pipeline diagram which illustrates, the 1st type, i.e., load youth stall, of a stall. The sequence of three instructions 610,614,618 exists in drawing 7 . The 1st instruction 610 is a CPU instruction as well as the 2nd instruction 614. The instruction by which a fetch is carried out to the 3rd is a floating point instruction. Instruction 610 is a load instruction which loads the contents of memory 146 to a register R1 in the address stored in the register R2. The 2nd instruction 614 is an instruction which adds the contents of the register R1 to the contents of the register R2. The 1st instruction 610 has accessed memory 146 on the CPU pipeline's 114 A stage 146, and the stall of the 2nd instruction 614 is carried out. Otherwise, the 2nd instruction 614 will access the contents by which a register R1 is not updated, while the instruction 610 has updated the contents of the register R1. If a stall is carried out, instruction 614 will redo processing from the decoding stage D.
[0051] Since the stall in one pipeline causes a stall on both pipelines (a CPU pipeline and FPU pipeline), the stall of the 3rd instruction 618 which is a floating point instruction will be carried out with the 2nd instruction 614. The CPU pipeline's 214 D stage 238 generates the stall signal 622 for carrying out the stall of both pipelines 210 and 214. In order to prevent an additional stall, the contents of the A stage 246 are bypassed by the CPU pipeline's 214 E stage 642. If this bypass is not performed, a stall must be carried out until the write back stage S of instruction 610 is completed.
[0052] Drawing 8 illustrates, another type, i.e., the memory access conflict stall, of a stall. The stall shown in drawing 8 happens, when two instructions tend to access cache memory 146 at coincidence. A pipeline diagram for drawing 8 to perform the sequence of four instructions is shown. The 1st instruction is the CPU MUBU instruction 751 as CPU MUBU instruction 610 of drawing 7 . Instruction 715 accesses memory 146 to the instruction of drawing 7 .
[0053] In drawing 8 , the 1st instruction 715 is one of two instructions by which the fetch was carried out in single fetch actuation in the instruction fetch stage 218, and by drawing 8 , since it is easy, the instruction fetch of another side is not illustrated. That is, it is because it enables it to perform the instruction fetch to a 16-bit fixed-length instruction per 2 instructions (32 bits) through a 32-bit data bus. In drawing 8 , when the preparation whose instruction fetch stage 218 fetches the 4th instruction from memory 146 will be completed, since the 1st instruction 715 accesses memory 146 on the A stage 246, the Di stage 238 carries out the stall of both pipes 210,214. Therefore, compared with the case where a stall is not carried out, 1 clock-cycle ****** fetch of the 4th instruction 720 is carried out.
[0054] Furthermore, the 3rd instruction 730 is repeated by application of the stall signal 1010 included in explanation of drawing 11 described further below for redo of Di stage. Furthermore, NOP (non, - operation) is inserted in the E stage 642 of the 3rd instruction 730 by application of the stall signal 725. And since carrying out the stall of one side will carry out the stall of another side similarly, the stall of both pipelines 210,214 is carried out.
[0055] Drawing 9 is the diagram of the circuit (it can set on the CPU pipeline's 214 Di stage 238) 810 which generates the stall signal 622,725. A comparator 815 is used in order to determine whether the destination register (610 instructionsR1 of drawing 7) of a circuit 810 is the same as that of a source register (614 instructionsR1 of drawing 7) in order to generate the stall signal 622 of drawing 7 . If there is such identity, OR circuit 820 will generate the stall signal 622.
[0056] Similarly, if the memory access circuit 825 determines the conflict of memory access, as described above about drawing 8 , OR circuit 820 will generate the stall signal 725 then.
[0057] Drawing 10 illustrates about how synchronization is maintained, when the CPU decoding stage 238 carries out the stall of the both sides of CPU and the FPU pipeline 210,214 to coincidence. In drawing 10 , two instructions are executed continuously. At first, the fetch of the CPU load communication register instruction explained by drawing 3 is done by FPU and the CPU pipeline 210,214, and it is executed. Next, the instruction fetch stage 218 fetches a floating point instruction 910. (The instruction fetch stage (I) 218 fetches two 32-bit instructions at once.) This fetch is produced in the even number word boundary. Therefore, it is not necessary to begin a fetch cycle for every instruction. A floating point instruction 910 interprets the contents of the register FPUL as an integral value. The floating point instruction 910 changes the integral value into a floating point number further. Finally, a floating point instruction 910 stores a floating point number in a floating-point register FRn.
[0058] drawing 10 -- setting -- a floating point instruction -- 910 -- both sides -- activation -- a stage (E1) -- 226 -- and -- (-- E --) -- 242 -- a stall -- a signal -- 914 -- using -- CPU -- 118 -- decoding -- a stage -- (-- Di --) -- 238 -- a result -- ****** -- FPU -- and -- CPU -- a pipeline -- 210,214 -- a stall -- carrying out -- making . That is, the stall of the activation stage 226,242 of both pipelines 210,214 is carried out. Such a stall is expressed in common by the alphabetic character "X" 918. This stall is suitable here and it

.is because the A stage 246 of CPU118 where it executes an instruction "LDS Rm, FPUL" when the activation preparation completion of the floating point instruction 910 will be carried out ordinarily has not yet made the data on a data bus 122 available. By carrying out 1 cycle stall, a floating point instruction 910 continues processing, just as the stall signal 914 is made into a low level. In the output of the A stage 246 of the CPU pipeline 214 who executes an instruction "LDS Rm, FPUL", available data is bypassed by the data bus 122 via the bypass path 922, and is bypassed by the input of E1 stage 226 of the FPU pipeline 210 who executes instruction 910 from there. It prevents that this bypass 922 adds a stall further. A floating point instruction 910 is because an instruction can be executed by processing the value of an available FPUL register on a data bus 122.

[0059] Similarly, in the mode of this operation, when pipeline Fries gets up on one pipeline, it causes Fries, both pipelines (FPU and CPU pipeline 210,214), to coincidence. All actuation in pipelines 210 and 214 stops between pipeline Fries. Fries, both pipelines 210,214, is produced as a result for example, of a cache mistake. A cache mistake is produced when CPU118 requires the data which does not exist in cache memory 146. In that case, the signal which shows that the cache mistake produced the cache controller 142 in CPU118 is sent. Moreover, when IMAC120 has not completed the multiplication processing whose CPU instruction requires a result, the IMAC120 produces pipeline Fries for a busy signal by delivery and it in CPU118 as well as FPU114.

[0060] Drawing 11 is the circuit diagram of the FPU pipeline's 210 decoding stage Df222. It depends for the function of the decoding stage (Df) 222 on the stall signal 1110 (refer to drawing 11) generated by the decoding stage (Di) 238 of CPU118. This stall signal 1110 is used on the FPU pipeline's 210 activation stage (E1) 226, and generates NOP on E1 stage 226. The Df stage's 222 detection of the conditions of a stall returns the Df stage 222 of drawing 11 in the instruction decoded by Df222 from the output 1014 of Df. The feedback through a path 1018 is attained by the control signal 1010 from the Di stage 238 for choosing the input 1006 (selector = 1) of a multiplexer 1008.

[0061] Drawing 11 shows the Fries signal selection path 1022 again. As mentioned above, the Fries signal stops all processings on all pipelines. Similarly, the Fries signal is applied to both pipelines 210,214, as mentioned above. The Fries signal 1022 disables latch 1024. Moreover, the cancellation signal selection path 1026 is shown in drawing 11 . Choosing a cancellation signal in the cancellation signal selection path 1026 is that a multiplexer (MUX) inserts NOP1030, and, thereby, it can cancel any instruction on the point (stage) of a pipeline. The portion shown by 1034 shows the signal further applied to the FPU pipeline's 210 decoding stage 222.

[0062] In description of the portion shown by 1034, explanation of the signal (for example, Signal A, a signal (B)) shown as a symbol is a C mark. In the portion shown by 1034, the vertical line in the definition of a signal name shows logical "OR" (OR). "&" shows logical "AND" (AND). "-" shows logical reversal.

[0063] Drawing 12 shows the diagram of the FPU pipeline's 210 1st activation stage (E1) 226. When a stall is suitable on the 1st activation stage 226, the decoding stage 238 of CPU118 applies a stall signal to the stall signal pass 1110 as mentioned above. This bars that 1038 is passed toEoutput of FPU decoding stage 222 1 stage. Instead, NOP1114 which should be inserted in E1 stage 226 is generated by asserting a stall signal in a path 1110. Application of the Fries and the cancellation signal in drawing 12 is the same as that of it of drawing 11 . Those signals are further explained in full detail by the portion shown by 1114.

[0064] drawing 13 -- FPU -- a pipeline -- 210 -- a stage -- (-- Df --) -- 222 -- 226 (E1) -- 230 (E2) -- and -- (-- Sf --) -- 234 -- being related -- being detailed -- a circuit -- a diagram -- it is . The FPU pipeline circuit 1208 shows some inputs. An input 1210 is for a first operand and an input 1214 is for the second operand of FPU or a CPU instruction. An input 1218 receives the bypass data for returning the output of E2 stage 230 to the input of E1 stage. An input 1222 is for bypassing the contents of the data bus (SD_2) 122 to the input of E1 stage 226. An input 1226 is an input from FR0 register of a floating point register file. Two or more latches to whom hatching is given partially, respectively operate [2 of a clock signal phases]. If based on the name described by above-mentioned explanation and drawing 13 , this contractor will be able to understand the remaining portion of a circuit 1208 easily.

[0065] As other classes of instruction sequence used for synchronization between FPU and the CPU pipeline 210,214, there is a floating-point-compare instruction which CPU branch instruction follows. Drawing 14 shows such a sequence. A floating-point-compare instruction sets the value of 1 to T bits, when the value of a floating-point register FRm1318 is equal to the value of a floating-point register FRn1322. The value of T bits determines whether the jump to a branching place is performed. Branch instruction 1314 is a CPU instruction. If T bits is 1, branch instruction produces the fetch of a new

instruction from a branching place (in the case of this example, it becomes with the fetch from cache memory 146). In drawing 14 , the value of T bits is bypassed by the CPU pipeline's 214 E stage 242 from 210EFPU pipeline 1 stage 226. If it puts in another way, while executing the floating-point-compare instruction, the output of T bits is performed on E1 stage. The bypass to such T-bit them prevents the stall in activation of the CPU branch instruction 1314 which continues after floating-point-compare ******. If T bits is outputted on E2 stage, it will be because the stall of 1 cycle is needed.

[0066] Drawing 15 shows the circuit for bypassing T bits from E1 stage 226 to the E stage 242. Furthermore, the circuit of drawing 15 has the capacity which bypasses T bits between a CPU pipeline's stages. When it is made to correspond to an example of the bypass of drawing 14 , are explained and T bits of values of 1410 are set to 1, the first instruction 1310, for example, floating-point-compare instruction, FPU activates T bit-select pass 1414, in order to choose T bits from E1 stage 226 through a multiplexer 1418. A selection circuitry 1422 can be chosen now from some possible T bit sources like a CPU pipeline's A stage latch 1426, or Si stage latch 1430 that it may be from a status register 1434. A selection signal 1432 is supplied from the decoding stage (Di) 238, and is based on the instruction under current activation on a pipeline stage. The address with which, as for the branch-address generating circuit 1438, CPU can fetch the next instruction is generated. Of course, if 1410 has T bits of values of 0 from E1 stage 226, the branch-address generating circuit 1438 will choose the address of the instruction which continues immediately after branch instruction in the sequence of a program. Since the target instruction is fetched, the address of the next instruction 1442 goes to the CPU pipeline's 214 E stage 242.

[0067] An instruction sometimes causes an exception. For example, it is a time of an instruction doing a division by 0 or using an illegal-instruction code. If an exception arises, an instruction sequence for an exception handler to process an exception generally will be performed. Then, an exception handler permits continuing the program execution accompanied by the instruction which caused the exception in CPU118. That is, the instruction which produced the exception is rerun. In the mode of this operation, the exception is exact in the semantics of not disturbing actual instruction-execution sequence to the instruction-execution sequence which a program specifies. For example, the exception of a floating point instruction is detected on 210EFPU pipeline 1 stage 226. When CPU118 maintains the sequence of the original program of a computer program, although an exception is an exception, it is exact. The sequence of an original program is the sequence of original instruction sequence as a compiled computer program. If it puts in another way, when an exception is exact, the instruction of a computer program will be executed as if they were performed by what does not have a continuous system, i.e., pipeline capacity, purely. An exception handler evacuates a condition of equipment like the condition of the processor 110 before an exception arises.

[0068] having a common stall and common Fries on both pipelines 210,214 -- the same -- an exception -- CPU and the FPU pipeline 210,214 -- setting -- both sides -- it is made exact by having the same number of pipe stages (pipelines' 210 and 214 synchronization). In order to attain synchronization of the stall in both pipelines 210,214, and Fries, the additional circuit for a long floating point instruction is included. Drawing 16 shows how when the floating point pipeline's 210 decoding stage 222 produces an exception, it keeps exact the execution sequence of an actual (as opposed to the instruction-execution sequence specified by description of a program) instruction by asserting a busy signal, even when a floating point instruction spends comparatively long time amount on completion. When a busy signal 124 is asserted (for example, high-level set), other floating point instructions cannot progress across the FPU pipeline's 210 Df stage 222 until E1 stage 226 is freed. While a busy signal is asserted, when the fetch of the CPU instruction is carried out, the instruction is executed using a CPU pipeline. On the other hand, if the fetch of other floating point instructions is carried out while the busy signal is asserted, the instruction concerned will be repeated on Df stage. The stall also of the CPU pipeline 214 who answers this is carried out.

[0069] In drawing 16 , the fetch of each instruction is carried out by the I stage 218 in the sequence shown by 1-6, and it is executed by the FPU pipeline 210 or the CPU pipeline 214. Drawing 16 illustrates the sequence of six instructions. The 1st instruction is the floating-point-divide instruction 1010. Once the FPU pipeline's 210 decoding stage Df222 decodes a floating point instruction 1510, the Df stage 222 will assert a busy signal 1508, and will supply it to the decoder of CPU. The next CPU instruction as shown by 2 and 3 after the busy signal 1508 is asserted by the busy signal path 124 then can continue activation. However, the next FPU instruction 1514 shown by 4, i.e., a floating-point-add instruction, is going to carry out the stall of the CPU pipeline 214 by application of the stall signal 1110 as mentioned above. Although the FPU pipeline stages (E1) 226 and 230 (E2) and 234 continue

activation of an instruction of 1, the new floating point instruction after the instruction of 1 is not allowed to progress previously from Df stage so that it may be represented by the instruction of 4. As for the FPU pipeline's 210 Df stage 222, termination of that the 1st floating point instruction is once executed by E1 stage negates a busy signal 1508. Transmitting 1514 toEfollowing floating-point-instruction, for example, floating-point-add instruction, 1 stage 226 is permitted by this.
[0070] An arrow head 1518 shows that a floating-point-add instruction goes to 210EFPU pipeline 1 stage which continues after 12 cycles which activation ofEfloating-point-divide instruction 1 stage of 1510 takes. An arrow head 1522 shows similarly that the 5th instruction, i.e., a CPU add instruction, goes to DI decoding stage 238 of the CPU pipeline 214 who continues after the 12th cycle in 1510Efloating-point-divide instruction 1 stage 226. Thus, when a floating point instruction like "FDIV" with the number of execution cycles longer than usual is executed, and a stall is appropriately carried out by said busy signal, a CPU pipeline and an FPU pipeline can maintain a synchronization. When an exception is generated by activation of the floating-point-add instruction 1514, the floating-point-add instruction 1514 makes an exception temporarily generated by 13 cycle eye 1526. Therefore, the exception does not disturb program execution sequence. That is, the exception is exact. It is because the next instruction of the floating-point-add instruction 1514 does not have the opportunity of activation yet. That is, when the floating-point-add instruction 1514 produces an exception by 13 cycle eye, it is because the instruction of consecutiveness has not gone to the activation stage yet. This is also because the CPU pipeline and the FPU pipeline are maintaining synchronization.
[0071] Drawing 17 shows the busy signal circuit for asserting a busy signal 1508 for the busy signal path 124. Especially the signal that shows the fact that the first instruction was inputted into the FPU pipeline's 210 decoding stage (Df) 222 is applied to logical AND gate (AND) 1618. Furthermore, the decoding stage (Df) 222 applies the first floating point instruction and the signal which shows the fact that the floating-point-divide instruction was inputted into E2 stage in the case of this example. However, before the signal is processed by AND gate 1618, it is reversed by the inverter 1626. and -- the beginning -- a floating point instruction -- E -- one -- a stage -- 226 -- inputting -- having had -- ** -- saying -- the fact -- being shown -- a signal -- **** ---like -- an OR gate -- (-- OR --) -- 1638 -- an input -- 1634 -- applying -- having . The output of AND gate 1618 is similarly applied to the input 1646 of another side of OR gate 1650. The output 1650 of OR gate 1638 can repeat an instruction on the FPU pipeline's 210 Df stage while it can supply a busy signal 1508 to the CPU pipeline's 214 decoding stage (Di) 238 and can make the CPU pipeline 214 generate a stall by it.
[0072] Although invention made by this invention person above was concretely explained based on the operation gestalt, it cannot be overemphasized that it can change variously in the range which this invention is not limited to it and does not deviate from the summary. For example, the pipeline number of stages of CPU and FPU is not limited to five steps, respectively, but can be changed suitably. Moreover, this invention is not limited to a RISC processor, but can apply a CISC processor etc. to the microprocessor which has other architecture. Moreover, in this specification, the microprocessor is used as a concept containing a microcomputer, a single chip microcomputer, and a data processor.
[0073]
[Effect of the Invention] It will be as follows if the effect acquired by the typical thing among invention indicated in this application is explained briefly.
[0074] That is, the microprocessor concerning this invention can make the resource for pipeline controls, such as a flow control of a CPU pipeline and an FPU pipeline, share by advancing an FPU pipeline and a CPU pipeline in juxtaposition, and considering as the pipeline number-of-stages identitas of a floating point pipeline and a CPU pipeline on the occasion of an instruction execution. Contraction of the circuit scale for pipeline control of share-izing of this resource is enabled. These make a floating point pipeline and a processor pipeline easy to synchronize.
[0075] When you need Fries and a stall to a pipeline, and a pipeline's one side controls a stall or Fries to bring both pipelines the effect of a stall and Fries, it can attain synchronization of an FPU pipeline and a CPU pipeline.
[0076] When CPU and FPU share a register like a FPUL register, the data exchange between CPU and FPU can be easily performed at a high speed.
[0077] The throughput to conditional branching which continues after a floating-point-compare instruction can be raised.
[0078] An FPU pipeline and a CPU pipeline can be synchronized also in a floating-point-divide instruction.

[Translation done.]

DESCRIPTION OF DRAWINGS

[Brief Description of the Drawings]
[Drawing 1] It is the block diagram showing an example of the microprocessor by this invention.
[Drawing 2] It is explanatory drawing showing the relation of the pipeline of FPU and CPU.
[Drawing 3] It is explanatory drawing showing the timing of the data exchange between the FPU pipeline for activation of an instruction "LDS Rm, FPUL", the pipeline of CPU, and both pipelines.
[Drawing 4] It is explanatory drawing showing the timing of the data exchange between the FPU pipeline for activation of an instruction "STS FPUL, Rn", the pipeline of CPU, and both pipelines.
[Drawing 5] It is explanatory drawing showing the pipeline of FPU in details further.
[Drawing 6] It is explanatory drawing showing a synchronous example of the FPU pipeline by sharing of a pipeline resource, and a CPU pipeline.
[Drawing 7] It is the pipeline diagram which illustrates the load youth stall which is the 1st gestalt of a stall.
[Drawing 8] It is the pipeline diagram which illustrates the memory access conflict stall which is another gestalt of a stall.
[Drawing 9] It is the block diagram which illustrates the circuit which generates a stall signal.
[Drawing 10] A CPU decoding stage is explanatory drawing which illustrates how a synchronization is maintained by carrying out the stall of the both sides of an FPU pipeline and a CPU pipeline.
[Drawing 11] It is explanatory drawing which illustrates an FPU pipeline's decoding stage.
[Drawing 12] It is explanatory drawing which illustrates an FPU pipeline's 1st activation stage E1.
[Drawing 13] It is the block diagram which illustrates the details circuit of FPU.
[Drawing 14] It is explanatory drawing which illustrates FPU for the procedure of a floating-point-compare instruction in which CPU branch instruction follows, and a CPU pipeline's synchronization.
[Drawing 15] It is the block diagram showing an example of the circuit for bypassing T bits on E stage from E1 stage.
[Drawing 16] It is explanatory drawing showing an example of synchronization of the pipeline of FPU and CPU for keeping an exception exact.
[Drawing 17] It is the block diagram showing an example of the busy signal circuit for asserting a busy signal for a busy signal path.
[Drawing 18] It is explanatory drawing of a register conflict.
[Drawing 19] It is explanatory drawing showing notionally an example of the resource share condition of a CPU pipeline and an FPU pipeline.
[Drawing 20] It is explanatory drawing which enumerated a floating point instruction and the CPU instructions relevant to a floating point instruction among the instruction sets which the microprocessor of drawing 1 supports.
[Drawing 21] It is explanatory drawing of a floating point instruction "FABS."
[Drawing 22] It is explanatory drawing of a floating point instruction "FADD."
[Drawing 23] It is explanatory drawing of the floating point instruction "FADD" following drawing 22 .
[Drawing 24] It is explanatory drawing of the floating point instruction "FADD" following drawing 23 .
[Drawing 25] It is explanatory drawing of a floating point instruction "FCMP."
[Drawing 26] It is explanatory drawing of the floating point instruction "FCMP" following drawing 25 .
[Drawing 27] It is explanatory drawing of the floating point instruction "FCMP" following drawing 26 .
[Drawing 28] It is explanatory drawing of a floating point instruction "FDIV."
[Drawing 29] It is explanatory drawing of the floating point instruction "FDIV" following drawing 28 .
[Drawing 30] It is explanatory drawing of a floating point instruction "FLDI0."

[Drawing 31] It is explanatory drawing of a floating point instruction "FLDI1."
[Drawing 32] It is explanatory drawing of a floating point instruction "FLDS."
[Drawing 33] It is explanatory drawing of a floating point instruction "FMUL."
[Drawing 34] It is explanatory drawing of the floating point instruction "FMUL" following drawing 33 .
[Drawing 35] It is explanatory drawing of a floating point instruction "FNEG."
[Drawing 36] It is explanatory drawing of a floating point instruction "FSQRT."
[Drawing 37] It is explanatory drawing of a floating point instruction "FSTS."
[Drawing 38] It is explanatory drawing of a floating point instruction "FSUB."
[Drawing 39] It is explanatory drawing of the floating point instruction "FSUB" following drawing 38 .
[Drawing 40] It is explanatory drawing of the floating point instruction "FSUB" following drawing 39 .
[Drawing 41] It is explanatory drawing of a floating point instruction "FTRC."
[Drawing 42] It is explanatory drawing of the floating point instruction "FTRC" following drawing 41 .
[Drawing 43] It is explanatory drawing of a floating point instruction "FTST."
[Drawing 44] It is explanatory drawing of a CPU instruction "LDS."
[Drawing 45] It is explanatory drawing of the CPU instruction "LDS" following drawing 44 .
[Drawing 46] It is explanatory drawing of a CPU instruction "STS."
[Drawing 47] It is explanatory drawing of the CPU instruction "STS" following drawing 46 .
[Drawing 48] It is explanatory drawing of a floating point instruction "FLOAT."
[Drawing 49] It is explanatory drawing of a floating point instruction "FMAC."
[Drawing 50] It is explanatory drawing of the floating point instruction "FMAC" following drawing 49 .

[Drawing 51] It is explanatory drawing of floating-point-instruction"FMOV.
[Drawing 52] It is explanatory drawing of floating-point-instruction"FMOV following drawing 51 .
[Description of Notations]
110 Microprocessor
114 FPU
118 CPU
210 FPU Pipeline
214 CPU Pipeline
218 Instruction Fetch Stage (I)
222 FPU Pipeline's Decoding Stage (Df)
226,230 An FPU pipeline's activation stage (E1, E2)
234 FPU Pipeline's Write Back Stage (Sf)
238 CPU Pipeline's Decoding Stage (Di)
242 CPU Pipeline's Activation Stage (E)
246 CPU Pipeline's Memory Access Stage (A)
250 CPU Pipeline's Write Back Stage (Si)
255 Bypass, Stall, Cancellation Path
FPUL Communication register
428 Bypass Signal from CPU Decoding Stage
518 Bypass Signal from CPU Decoder
622 Stall Signal
725 Stall Signal
914 Stall Signal
1410 T Bits from E1 Stage
1508 Busy Signal
1608 Busy Signal

[Translation done.]